A Parser for Question-answer System for Tamil

Rajendran Sankaravelayuthan[1], M. Anandkumar[2], V. Dhanalakshmi[3], Mohan Raj S.N.[4]
1. Amrita Vishwa Vidyapeetham, Coimbatore, rajushush@gmail.com, 2. NITK, Surathkal, manandinfo@gmail.com, 3. Govt. Arts College for Women, Krishnagiri, dhanagiri@gmail.com,
4. Amrita Vishwa Vidyapeetham, Coimbatore, snmohanraj@gmail.com

Abstract: The present paper aims to describe about the parser and a question-answer system we propose to build for Tamil. We have almost completed building of a parser for Tamil. We have made use of dependency grammar for the creation of parser for Tamil. We have made use of the Stanford parser modal to meet our mission. The Stanford parser is judged as a reliable parser for NLP applications. More overs as the dependency relations such as subject, object, indirect object and other relations of the participants of the verb are taken care of in the dependency parser it is a very useful approach for our purpose. We have made use of the POS tag sets, chunk sets and dependency sets of Stanford parser by modifying them to suit our purpose. We have made use of machine learning technique to prepare dependency parser for Tamil. Our question answer system is domain restricted and works on structured database. We are making use of the already available 50 thousand sentences in Tourist domain to find answer for the questions queried by the user.

1. Introduction

Though a number of syntactic parser is attempted for Tamil, a full-fledged syntactic parser is yet to be made. More over a highly workable dependency parser for Tamil for NLP application is the need of the day. One such attempt is made here. Machine learning approach is made use of for building dependency parser for Tamil. We have made use of Stanford parser as the modal for our parser. The Stanford POS tag sets, chunk sets and dependency tag sets are adopted with minor modification for Tamil. Parsing is followed by question-answer system. We have fifty thousand Sentences in Tamil in tourist domain. This has been made while creating English-Tamil parallel corpus for English Tamil MT system. These sentences will be parsed by the parser developed by us and kept in the data base. The question asked will be parsed by making use of the same parser. For the given question the correct answer will be extracted from the data base containing parsed out of fifty thousand sentences.

2. Parser

A natural language parser is a program that works out the grammatical structure of sentences. This groups words together (as "phrases") that could be considered as subject, object, indirect object or any other arguments (participants) of a verb. A number of syntactic parser has been developed for Tamil [8], [17] [24]. So far a full-fledged syntactic parser is yet to be developed. Chunker has been developed for Tamil to meet the requirement of MT systems. Kumara Shanmugam [17] developed a syntactic parser for Tamil. It has its own limitations. The parser at par with Stanford parser for English should be targeted for Tamil. Stanford Parser (SP) is a well-known and highly dependable parser for English. It gives only one output for a given sentence. It is a kind of decency parser and it provides dependency information which is very useful for any kind of NLP applications. It is a probabilistic parser which uses knowledge of the language gained from hand-parsed sentences and to try to produce the most likely analysis of new sentences. The statistical parser still makes some mistakes, but commonly works rather well. It is claimed that its development was one of the biggest breakthroughs in natural language processing in the 1990s. It is available for general use in online. We are adopting its POS tag sets, chunk sets, and dependency tag sets for our purpose my modifying them to suit our requirement.

2. Creation of structured domain restricted text

We are not going to develop a full-fledged general question answer retrieval system. It is not going to be an open-ended system. The domain is restricted to tourist domain. We have developed a tourist document consisting of 50 thousand sentences as an outcome of creation of parallel corpus between English and Tamil. This is going to be our domain restricted structured text.

3. Building syntactic parser using Stanford parser labels

There are two steps in the system. First the questions will be processed by making use of the syntactic parser for Tamil. We will match the questions with the parsed output of the 50000 sentences in the data base of the tourist domain and select the correct answer. Named entity recognizer and co-reference recognizer will be plugged in to the parsing system.

For the given sentence the SP gives the tagged out put. This is be followed by parsing. Then universal dependencies are given. This is followed by enhanced universal dependencies. For example, the input sentence '*John hit the ball with the bat.'* is parsed as follows:

Your query

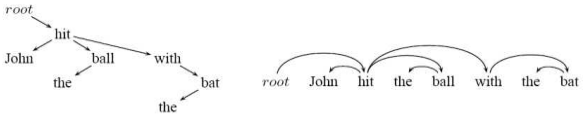
John hit the ball with the bat.

Tagging
John/NNP
hit/VBD
the/DT
ball/NN
with/IN
the/DT
bat/NN./.

Parse
(ROOT
(S
(NP (NNP John))
(VP (VBD hit)
(NP (DT the) (NN ball))
(PP (IN with)
(NP (DT the) (NN bat))))
(. .)))

Universal dependencies
nsubj(hit-2, John-1)
root(ROOT-0, hit-2)
det(ball-4, the-3)
dobj(hit-2, ball-4)
case(bat-7, with-5)
det(bat-7, the-6)
nmod(hit-2, bat-7)

This can be diagrammatically represented as follows.

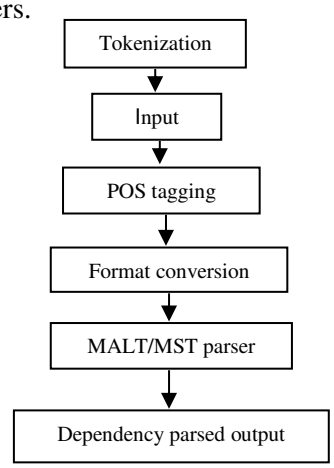


4. Machine learning approach to dependency parsing

Dependency parsing is one of the most important natural language processing works. Dependency parsing is the task of finding the dependency structure of a sentence. Dependency structure for a sentence is a directed graph originating out of a unique and artificially inserted root node. It is very important for machine translation and information extraction. The dependency parser helps to serve various NLP works: Relation Extraction, Machine Translation, Synonym Generation, Lexical Resource Augmentation, and Information Extraction.

The dependency parsing developed can serve the above requirement. The dependency parsing is can be implemented using rule based approach, but as it does not give the required result, an attempt is made here to full-fil our mission using machine learning approach. The machine learning approach tries to resolve the problems obtained in rule based approach. In rule based approach, a vast amount of linguistic knowledge is required, whereas in machine learning approach only a reasonable amount of the knowledge is required. In rule based approach, if one rule fails we need to make changes in all others. In the machine learning approach, two kinds of algorithm are used: one is supervised algorithm and another is unsupervised algorithm. Here supervised learning is used for creating the model and using MST Parser Tool and MALT parser Tool, the dependency labels and the position of the head are obtained. For viewing the dependency tree structure Graphic tool is used. The results obtained by both the tools have been compared. The results obtained are very encouraging.

For Tamil parsing, data driven dependency parsers (MALT and MST) are applied for identifying the dependency graph. The general framework of Tamil dependency parsing is illustrated in figure given below. This framework shows how the dependency structures for Tamil are identified using MALT and MST parsers.



The tokenized input sentence is fed to the POS tagger module, which is the primary process in parsing. The POS tagged sentences are given to Chunker module. The processed sentences are converted into the required format for Malt/MST parser. A PERL program is used for this conversion process.

4.1 POS Tagging

Parts of speech (POS) tagging is assigning grammatical classes i.e. appropriate parts of speech tags to each word in a natural language sentence. The POS tagging here is done using machine learning which makes it simple. There are two kinds of machine learning, one is supervised and the other is unsupervised learning. Here we use a supervised method of learning. In this method, we require pre-tagged Part of speech corpus to learn information about the tag set, word-tag frequencies, rule sets etc. The accuracy of the models generally increases with the increase in size of the corpus. Support Vector Machine (SVM) which is one of the powerful machine-learning methods is used here. A model is created for a considerable size of Tamil data and using this trained model the untagged data are tested. The SVM Tool for POS tagging gives laudable accuracy. For getting dependency parsing output, we need POS tagged structure. We make use of the following tag set adopted from Stanford parser for POS-tagging which is tabulated below.

S.N	TAG	DESCRIPTION		S.N	TAG	DESCRIPTION
1	<NN>	NOUN		16	<CNJ>	CONJUNCTION
2	<NNC>	COMPOUND NOUN		17	<CVB>	CONDITIONAL VERB
3	<NNP>	PROPER NOUN		18	<QW>	QUESTION WORDS
4	<NNPC>	COMPOUND PROPER NOUN		19	<COM>	COMPLIMENTIZER
5	<ORD>	ORDINALS		20	<NNQ>	QUANTITY NOUN
6	<CRD>	CARDINALS		21	<PPO>	POSTPOSITIONS
7	<PRP>	PRONOUN		22	<DET>	DETERMINERS
8	<ADJ>	ADJECTIVE		23	<INT>	INTENSIFIER
9	<ADV>	ADVERB		24	<ECH>	ECHO WORDS
10	<VNAJ>	VERB NON FINITE ADJECTIVE		25	<EMP>	EMPHASIS
11	<VNAV>	VERB NON FINITE ADVERB		26	<COMM>	COMMA
12	<VBG>	VERBAL GERUND		27	<DOT>	DOT
13	<VF>	VERB FINITE		28	<QM>	QUSTION MARK
14	<VAX>	AUXILIARY VERB		29	<RDW>	REDUPLICATION WORDS
15	<VINT>	VERB INFINITE				

Explanation of POS tags:

NN (noun): The tag NN is used for common nouns without differentiating them based on the grammatical information.

3. nalla kuzandai <NN>
'good child'

NNC (compound noun): Nouns that are compound can be tagged using the tag NNC.

4. kaN <NNC> vali <NNC>
'eye pain'

NNP (Proper Nouns): The tag NNP tags the proper nouns.

5. jaaN<NNP> angke niR-kiR-aan.
John there stand-PRES-3MS
'John is standing there'

NNPC (Compound Proper Nouns); Compound proper nouns are tagged using the tag NNPC.

6. aTal <NNPC> pikaari <NNPC> vaajpaayi <NNPC>
'Atal Bihari Vajpayi'

ORD (Ordinal): Expressions denoting ordinals will be tagged as ORD.

iraNTaavatu <ORD> kuzandtai.
'Second child'

CRD (Cardinal): Cardinal tag tags the cardinals like *onRu* 'one', iraNTu 'two', *muunRu* 'three' etc in the language as CRD.

7. iraNTu <CRD> puttakangkaL
'two books'

PRP (Pronoun): All pronouns are tagged using the tag PRP.

8. en <PRP> viiTu
'my house'

ADJ (Adjective): All adjectives in the language will be tagged as ADJ.

9. azakaana <ADJ> paaTTu
'beautiful song'

ADV (Adverb): Adverbial tag marks the adverbs in the language as ADV.

10. avan veekamaaka <ADV> ooT-i-kkoNTirukkiR-aan.
He fast run-PRES-CONT-3MS
'He is running fast'

VNAJ (Verb Non-finite Adjective): Non-finite adjectival forms of the verbs are tagged as VNAJ.

11. va-ndt-a <VNAJ> paiyan
come-PAST-ADJ boy
'the boy who came'

VNAV (Verb Nonfinite Adverb): Non-finite adverbial forms of the verbs are tagged as VNAV.

12. tuungk-i <VNAV>.ezu-ndt-aan
come-PAST-ADV go-PAST-3MS
'having come went'

VBG (Verbal Gerund): All gerundival forms of the verbs are tagged as VBG.

13. avan varu-kiR-atu <VBG>
he come-PRES-NOM
'that he is comming'

VF (Verb Finite): VF tag is used to tag the finite forms of the verbs in the language.

14. avan teervu ezhut-in-aan <VF>.
He exam write-PAST-3MS
'He wrote the examination'

VAX (Auxiliary Verb) : VAX tag is used to tag the auxiliary verbs in the language.

15. avan teervu ezut-i-koNTirikkiR-aan <VAX>.
He exam write-VNAV-CONTINUOUS ASPECT-3MS
'He is writing the examination'

VINT (Verb Infinite): The infinitive forms of the verbs are tagged as VINT in the language.

16. avan enn-aik kaaNa <VINT> va-ndt-aan
he I-ACC see-VINT come-PAST-3MS
'He came to see me'

CNJ (Conjuncts, both co-ordinating and subordinating): The tag CNJ can be used for tagging co-ordinating and subordinating conjuncts.

17. raaman-um kaNNan-um maRRum <CNJ> palar-um ingkee <ADV> irukkiR-aarkaL.
Raman-CNJ Kannan-CNJ CNJ many persons-CNJ here be-PAST3PHP
'Raman, Kannan and others are present here'

18. avan illaaViTTaal <CNJ> avaL varu-v-aaL
he or-CNJ she come-FUT-3FS
'He or she will come'

CVB (Conditional Verb): The conditional forms of the verbs are tagged as CVB.

19. avan-ai kaN-T-aal <CVB> pootum.
He-ACC see-CON enough
'It is enough to see him'

QW (Question Words): The question words in the language like *yaar* 'who', *endtaa* 'which' etc are tagged as QW.

20. yaar <QW> va-ndt-atu?
Who come-PAST-NOM
'Who came'

COM (Complimentizer): Complementizers are tagged as COM in the language.

21. avan avaL varu-v-aaL enRu <COM> kuuR-in-aan

He come-FUT-3FS that say-PAST-3MS
'He said that she would come'

NNQ (Quantity Noun): Quantitative nouns are tagged as NNQ in the language.

22. enakku konjca <NNQ> neeram taa.
me some time give
'Give me some time'

PPO (Postposition): All the Indian languages including Tamil have the phenomenon of postpositions. Postpositions are tagged using the tag PPO.

23. naan viiTu varai <PPO> varu-v-een.
I house upto come-FUT-1S
'I shall come up to the house'

DET (Determiners): The determiners in the language are tagged as DET.

24. andta <DET> kuzandtai
'that child'

INT (Intensifier) : Intensifier is used for intensifying adjectives or adverbs in a language. They are tagged as INT.

25. mika<INT> nalla kuzandtai
'Very good child'

ECH (Echo words): Echo words are common in Tamil language. They are tagged as ECH.

26. puli kili <ECH>
'tiger or that which resembles tiger'

EMP (Emphasis): The emphatic words in the language are tagged as EMP.

27. naan maTTum<EMP> varu-v-een.
'I only come-FUT-1S'
'Only I will come.'

COMM: The tag COMM tags the comma in a sentence.

28. cennai ,<COMM> tanjcaavuur , <COMM> maturai.
'Chennai, Thanjavur, Madurai'

DOT: The tag DOT tags the dots in the sentences.

29. naan angkee poo-n-een .<DOT>
I there go-PAST-1S
'I went there'

QM (Question Mark): The question marks in the language are tagged using the tag QM.

30. nii engkee poo-n-aay? <QM>
you where go-PAST-2S
'Where did you go?'

RDW (Reduplication Words): The reduplicated words are tagged as RDW.

31. mella mella <RDW>
'slowly slowly'

4.1.1 Training Corpus

The training data for POS-tagging is a two column format. First column contains input sentence and second column contains the POS tagged output. The following are the two examples.

32. avan oru kuTai vaangk-in-aan. he one umbrella buy-PAST-3MS 'He bought an umbrella.'	33. avan piyaanoo vaaci-kkiR-aan. he piyaanoo paly-PRES-3MS 'He is playing piano.'
avan <PRP> oru <DET> kuTai <NN> vaangkinaan <VF> . <DOT>	avan <PRP> piyaanoo <NN> vaacikkiRaan <VF> . <DOT>

A Model is created using this corpus data will be used for testing.

4.1.2. Testing sentences

The input sentences are aligned in a column manner using a program and then given to the SVM Tool for POS-tagging. Using the trained model the input sentences are tagged. The following is an example.

34. naan oru pazam paRikk-a mara-ttil eeR-in-een.
I one fruit pluck-PAST-INF tree-LOC claim-PAST-1S
'I climbed the tree to pluck a fruit'

Alignment Output (POS Tagging Input):	POS Tagged Output:
naan oru pazam paRikka marattil eeRineen.	naan <PRP> oru <DET> pazam <NN> paRikka <VINT> marattil <NN> eeRineen <VF> . <DOT>

4.2. Chunking

Chunking is an efficient and robust method for identifying short phrases in text, or chunks. The notion of phrase chunking is proposed by Abney. Chunking is considered as an intermediate step towards full parsing. After POS-tagging, the next step is chunking, which divides sentences into non-recursive inseparable phrases. A chunker finds adjacent, non-overlapping spans of related tokens and groups them together into chunks. Chunkers often operate on tagged texts, and use the tags to make chunking decisions. A subsequent step after tagging focuses on the identification of basic structural relations between groups of words. This is usually referred to as phrase chunking.

Chunking is comparatively easier for Indian languages than POS-tagging. The output of POS tagger is the input to the chunker. Chunking has been traditionally defined as the process of forming group of words based on local information. Hence, identifying the POS-tags and chunk-tags for the words in a given text is an important aspect in any language processing task. Both are important intermediate steps for full parsing. The word chunking tells something about how it is used for identifying short phrases or chunks in a text. Chunks are non-overlapping spans of text, usually consisting of a head (such as a noun) and the adjacent modifiers and function words (such as adjectives and determiners). A typical chunk consists of a single content word surrounded by a constellation of function words. Chunks are normally taken to be a non-recursive correlated group of words. Tamil being an agglutinative language has a complex morphological and syntactical structure. It is a relatively free word order language, but in the phrasal and clausal construction it behaves like a fixed word order language. So the process of chunking in Tamil is less complex compared to the process of POS-tagging. We followed the guidelines mentioned in AnnCorra: Annotating Corpora Guidelines for POS and chunk annotation for Indian Languages while creating our tag set for chunking. Our customized tag set contains ten tags and is tabled below.

S.No.	Tag	Tag name	Possible POS Tags
1	NP	Noun Phrase	NN, NNP, NNPC, NNC, NNQ, PRP, INT, DET, CRD, ADJ, ORD
2	AJP	Adjectival Phrase	CRD,ADJ
3	AVP	Adverbial Phrase	ADV, INT, CRD
4	VFP	Verb Finite Phrase	VF, VAX
5	VNP	Verb Nonfinite Phrase	VNAJ, VNAV, VINT, CVB
6	VGP	Verb Gerund Phrase	VBG
7	CJP	Conjunctional	CNJ
8	COMP	Complimentizer	COM
9	PP	Post Position	PPO,NN
10	.?	Symbols	O

4.3. Dependency Parsing

Parsing is actually related to the automatic analysis of texts according to a grammar. Technically, it is used to refer to the practice of assigning syntactic structure to a text. It is usually performed after basic morphosyntactic categories have been identified in a text. Based on different grammars parsing brings these morphosyntactic categories into higher-level syntactic relationships

with one another. The dependency structure of a sentence is defined by using dependency labels and dependency head. The following is the table for dependency tags used in the present system.

S.No.	Tag	Description
1	<ROOT>	Head Word
2	<NSUB>	Nominal Subject
3	<DOBJ>	Direct Object
4	<IOBJ>	Indirect Object
5	<.MOD>	Modifier
6	<CLSUB>	Clausal Subject
7	<CLDOBJ>	Clausal Direct Object
8	<CLIOBJ>	Clausal Indirect Object
9	<SYM>	Symbols
10	<X>	Others
11	<XCL>	Clause Boundary

Explanation of the Tag set

Noun Phrase (NP)

Noun Chunks will be given the tag NP. It includes non-recursive noun phrases and postpositional phrases. The head of a noun chunk would be a noun. Noun qualifiers like adjective, quantifiers, determiners will form the left side boundary for a noun chunk and the head noun will mark the right side boundary for it. Example for NP chunk is given below:

35.avaL oru azakaana peN.
she one beautiful woman
'She is a beautiful woman'

avaL	<PRP>	<B-NP>
oru	<DET>	<B-NP>
azakaana	<ADJ>	<I-NP>
peN	<NN>	<I-NP>
.	<DOT>	<O>

Adjectival Phrase (AJP)

An adjectival chunk is tagged as AJP**.** This chunk will consist of all adjectival chunks including the predicative adjectives. However, adjectives appearing before a noun will be grouped together with the noun chunk. It can be seen from the example for the noun phrase. Example for ADJ Phrase is given below:

36. palavakai aaraayccikkaaka ndaaTukaL tuNaikkooLkaL-ai viNveLi-kku anuppu-kinR-ana.
Different-type-ADJ research-ADV countries satellites-ACC space-DAT send-PAST-3NP
'The countries are sending satellites to space for different types of research works.'

palavakai	<ADJ>	<B-AJP>
aaraayccikkaaka	<ADV>	<B-ADP>
ndaaTukaL	<NN>	<B-NP>
tuNaikkooLkaLai	<NN>	<B-NP>
viNveLikku	<NN>	<B-NP>
anuppukinRana	<VF>	<B-VFP>
.	<DOT>	<O>

Adverbial Phrase (ADP)

Adverbial chunk is tagged in accordance with the tags used for POS tagging. It is tagged as AVP**.** An example for ADP is given below.

37. ndaan inRu cennai-kkup pookiReen.
I today Chennai go-PRES-1S
'I am going to Chennai today.'

ndaan	<PRP>	<B-NP>
inRu	<ADV>	<B-AVP>
cennaikkup	<NNP>	<B-NP>
pookiReen	<VF>	<B-VFP>
.	<DOT>	<O>

Conjunction

Conjunctions are the words used to join individual words, phrases, and independent clauses. The conjunctions are labelled as CJP. An example is given below.

38. raaNikku veLLimeTal kiTai-tt-atu enRaalum tangkameTal kiTaikkavillai
Rani-DAT silver medal get-PAST-3NS CNJ gold medal get-not
'Though Shaini got silver medal, she did not get gold medal'

raNikku	<NNP>	<B-NP>
veLLimeTal	<NN>	<B-NP>
kiTaittatu	<VF>	<B-VFP>
enRaalum	<CNJ>	<B-CJP>
tangkameTal	<NN>	<B-NP>
kiTaikkavillai	<VF>	<B-VFP>
.	<DOT>	<O>

Complimentizer

Complimentizers are the words equivalent to the term subordinating conjunction in traditional grammar. For example, the word *that* is generally called a Complimentizer in English. Complimentizer is tagged in accordance with the tags used for POS tagging. It is tagged as COMP.

39. uttaravai maaRRivaikk-a muTiy-aatu enRu vazakkaRinjar con-n-aar.
order postpone-INF possible-not COM advocate say-PAST
'The advocate said that the order cannot be postponed.'

uttaravai	<NN>	<B-NP>
maaRRivaikka	<VINT>	<B-VNP>
muTiyaatu	<VAUX>	<B-VFP>
enRu	<COM>	<B-COMP>
vazakkaRinjar	<NN>	<B-NP>
con-n-aar	<VF>	<B-VFP>
.	<DOT>	<O>

Verb Finite Phrase (VFP)

Verb chunks are mainly classified into verb finite chunk and verb non-finite chunk. Verb finite chunk includes main verb and its auxiliaries. It is tagged as VFP. An example of VFP chunk is given below.

40. avarkaL veelainiRuttam ceyy-a avan-ai azai-tt-anar.
They strike conduct-INF they-ACC invite-PAST-3HP
'They invited him to conduct the strike.'

avarkaL	<PRP>	<B-NP
veelainiRuttam	<NN>	<B-NP>
ceyya	<VINT>	<B-VNP>
avanai	<PRP>	<B-NP>
azaittanar	<VF>	<VFP>
.	<DOT>	<O>

Verb Non-finite Phrase (VNP)

Non-finite verb comprise all the non-finite form of verbs. There are four non-finite forms in Tamil and they are relative participle form, adverbial participle form, conditional form and infinitive form. They are tagged as VNP. An example of VNP chunk is given below.

41. avaL 58 nimiSa-ttil ooT-i ett-in-aaL.
She 58 minutes-LOC run-VNAV reach-PAST-3FS
'He reached by running in 58 minutes'

avaL	<PRP>	<B-NP>
58	<CRD>	<B-NP>
nimiSattil	<NN>	<I-NP>
ooTi	<VNAV>	<B-VNP>
ettinaaL	<VF>	<B-VFP>
.	<DOT>	<o>

Verb Gerundial Phrase (VGP)

Gerundial forms are represented by a separate chunk. They are tagged as VGP. An example of VGP chunk is given below.

42. avanukku paTam varai-v-at-il atika viruppam uNTu.
    he picture draw-PRE-NOM-LOC more interest is
    'He has more interest in drawing pictures'

avanukku	<PRP>	<B-NP>
paTam	<NN>	<B-NP>
varaivatil	<VBG>	<B-VGP>
atika	<ADJ>	<B-NP>
viruppam	<NN>	<I-NP>
uNTu	<VAX>	<B-VFP>
.	<DOT>	<O>

Symbol (O)

Special characters like Dot (.) and question mark (?) are tagged as O. Comma is tagged with the preceding tag.

43. aRaiy-il meecaikaLoo naaRkaalikaLoo illaatataal peritaakat teri-kiR-atu
    room-LOC tables chairs not-having big-ADV appear-PRES-3NS
    'As there are no tables and chairs in the room it appears big.'

aRaiyil	<NN>	<B-NP>
meecaikaLoo	<NN>	<B-NP>
naaRkaalikaLoo	<NN>	<B-NP>
illaatataal	<VNAV>	<B-VNP>
peritaaka	<ADV>	<B-AVP>
tooRRukiRatu	<VF>	<B-VFP>
.	<DOT>	<O>

3.3.1. Dependency Head and Dependency Relation

The parent child relation is specified using the arc. These arcs are symbolically represented using the position of the parent i.e. the number; this is explained below using an example.

44. raaman kaNNan-ukku oru pazam koTu-tt-aan.
    Raman Kannan-DAT one fruit give-PAST-3MS
    'Raman gave a fruit to Kannan'

1	2	3	4	5	6
raaman	kaNNanukku	oru	pazam	koTuttaan	.
<NNP>	< NNP>	<DET>	<NN>	<VF>	<DOT>

1	raaman	5	<NNP>	<NSUB>
2	kaNNanukku	5	<NNP>	<IOBJ>
3	oru	4	<DET>	<X>
4	pazam	5	<NN>	<DOBJ>
5	koTuttaan	0	<VF>	<ROOT>
6	.	5	<DOT>	<SYM>

This can be explained in the following way: *raaman* is the child of the parent *koTuttaan* which is in position 5; *kaNNanukku* is the child of the parent *koTuttaan* which is in the position 5; *oru* is the child of the parent *pazam* which is in position 4; *pazam* is the child of the parent *koTuttaan* which is in the position 5. *koTuttaan* which is the ROOT is in position 5; "." is the child of the parent *koTuttaan* which is in position 5. The children are linked to the parent by arcs and the arcs are labelled accordingly as NSUB, IOBJ, X, DOBJ, ROOT and SYM.

4.3.2. MALT Tool

The tool used for Dependency Parsing is the MALT Parser Tool. MALT stands for Models and Algorithms for Language Technology. It has been developed by Johan Hall, Jens Nilsson and Joakim Nivre at the Vaxjo University and Uppsala University of Sweden. MALT Parser is a

system for data-driven dependency parsing. The parser can be used to induce a parsing model from the training data and to parse new data using the induced model. The parser uses the transition based approach to parse the new data. Transition based parsing builds on idea that parsing can be viewed as a sequence of state transitions between states and this approach uses a greedy algorithm. Parsers built using MALT Parser have achieved a high state-of-the-art accuracy for a number of languages.

There are 10 features in the MALT parser. The ten features are listed as follows: 1. Word Index, 2. Word, 3. Lemma, 4. Coarse Parts of Speech tag, 5. Parts of Speech Tag, 6. Morphosyntactic Features, 7. Dependency Head, 8. Dependency Relation, 9. Phrasal Head, 10. Phrasal Dependency Relation. These features are user defined for the training data and the features which are not defined are marked null represented with the symbol "_". In our model, we have considered the following features: 1. Word Index, 2. Word 3. POS tag, 4. Chunk Tag, 5. Dependency Head, 6. Dependency Relation, The rest of the features is marked „_".

4.3.2.1. Training

The system is trained with more than 10,000 data which contains around 2000 sentences each of different patterns. This covers almost all the patterns for simple sentences and complex sentences of smaller length. The training data has the word id, word, pos tag, chunk tag, dependency head and dependency relation. The other columns are null and are denoted by an „_" (Underscore). The training data format is given below.

ID1	W1	P1	C1	_	_	H1	D1	_	_
ID2	W2	P2	C2	_	_	H2	D2	_	_
ID3	W3	P3	C3	_	_	H3	D3	_	_
.									
.									
.									
IDn	Wn	Pn	Cn	_	_	Hn	Dn	_	_

Here, ID refers to word index, W refers to word, P refers to parts of speech tag, C refers, to chunk tag, H refers to dependency read, and D refers to dependency relation. With this training data format, a model is developed. An example of the training data is given below.

45. raaman oru azaiipitaz koTuttuviTTu avan-ai viiTT-ukku azai-tt-aan

Raman one banana give-VNAV he-ACC house-DAT invite-PAST-3MS

'Having given banana to him, Raman invited him to the house.'

1	raaman	<NNP>	<B-NP>	-	-	4	<CLIOBJ>	-	-
2	oru	<DET>	<B-NP>	-	-	3	<X>	-	-
3	azaippitaz	<NN>	<I-NP>	-	-	4	<CLDOBJ>	-	-
4	koTuttuviTTu	<VNAV>	<B-VNP>	-	-	7	<XCL>	-	-
5	avanai	<PRP	<B-NP>	-	-	7	<DOBJ>	-	-
6	viiTTukku	<NP>	<B-NP>	-	-	7	<X>	-	-
7	azaittaan	<VF>	<B-VFP>	-	-	0	<ROOT>	-	-
8	.	<DOT>	<O>-	-	-	7	<SYM>	-	-

4.3.2.2. Testing

The test data to be given to the malt parser has the Word ID, Word, pos tag and the Chunk tag. The remaining columns are given null. Testing input format is given below.

ID1	W1	P1	C1	_	_	_	_	_	_
ID2	W2	P2	C2	_	_	_	_	_	_

In the testing input format the head and dependency relation are given as NULL. The output data for the above test input data will be as follows.

ID1	W1	P1	C1	_	_	H1	D1	_	_

The head and dependency relation are obtained. It should also be noted that the training and the test data should have same features. An example of the input test data is given below.

46. raaman kaNNanukku uNavu koTuttuviTTu poonaan
    Raman Kannan-DAT food give-VNAV go-PAST-3MS
    'Having given Kannan the food, he went away'

1	raaman	<NNP>	<B-NP>	-	-	-	-	-
2	kaNNanukku	<NNP>	<B-NP	-	-	-	-	
3	uNavu	<NPN>	<B-NP>	-	-	-	-	-
4	koTuttuviTTu	<VNAV>	<B-VNP>	-	-	-	-	-
5	poonaan	<VF>	<B-VFP>	-	-	-	-	-
6	.	<DOT>	<O>	-	-	-	-	-

The output for the test data is as follows.

1	raaman	<NNP>	<B-NP>	-	-	5	<N.SUB>	- -
2	kaNNanukku	<NNP>	<B-NP	-	-	4	<CL.IOBJ>	- -
3	uNavu	<NPN>	<B-NP>	-	-	4	<CL.IOBJ>	- -
4	koTuttuviTTu	<VNAV>	<B-VNP>	-	-	5	<X.CL>	- -
5	poonaan	<VF>	<B-VFP>	-	-	0	<ROOT>	- -
6	.	<DOT>	<O>	-	-	5	<SYM>	- -

In the test input data, the Dependency Head and Dependency Relation are given as NULL.

4.3.2.3. Learning Algorithm

The version of MALT used for training and testing is MALT 1.4.1. MALT Parser uses the Shift Reduce Parser for parsing the data. The arc labels are identified using the LIBSVM classifier. The MALT parser has two algorithms for classifying the tags. One is LIBSVM and the other learning algorithm is LIBLINEAR. LIBSVM uses the Support Vector Machines and LIBLINEAR uses various linear Classifiers. Both the learning algorithm has its own advantages and disadvantages. The developed model for parsing uses the LIBSVM for classifying the parse data.

4.3.3. MST Parser Tool

MST Parser Tool is also another machine learning tool which uses supervised learning algorithm. Using this tool dependency labels and position of each word parent are obtained.

4.3.3.1 Training

A corpus is created for training in the format given below.

47. raaman oru paamp-aik kon-R-aan.
    Raman one snake-ACC kill-PAST-3MS
    'Raman killed a snake'

raamu	oru	paamp-aik	konRaan	.
NNP	DET	NN	VF	.
NSUB	X	DOBJ	ROOT	SYM
4	3	4	0	4

48. pacu pul tin-kinR-atu
    cow grass eat-PRES-3MS
    'The cow is eating grass'

pacu	pul	tinkiRatu	.
NN	NN	VF	.
N.SUB	D.OBJ	ROOT	SYM
3	3	0	3

49. avan oru kappu koNTuva-ndt-aan.
    He one stick bring-PAST-3MS
    'He brought a stick.'

avan	oru	kappu	koNTuvandtaan
PRP	DET	NN	VF .
N.SUB	DET	D.OBJ	ROOT SYM
4	3	4	0 4

50. ciitai oru caari vaangk-in-aaL
    Sita one sari buy-PAST-3FS
    'Sita bought a sari'

ciitai	oru	caari	vaangkinaaL	
NN	DET	NN	VF	.
N.SUB	DET	D.OBJ	ROOT	SYM
4	3	4	0	4

In this way a corpus is created for Malayalam language. This corpus is trained using the MST Parser Tool and thus a model is created. Using this model as source the new inputs are tested.

4.3.3.2. Testing

POS-tagged Output is the input for MST Parser Tool. The above pos-tagged output is converted into the MST input format using a Perl program. This is then given to MST Parser Tool and by using the trained model, required output is obtained, i.e. position of each word parent and dependency labels of the Pos-tagged Sentence are obtained. An example is given below.

51. naan oru pazam paRikka marattil eeR-in-een.

I one fruit pluck-VINT tree-LOC climb-PAST-1S

'I climbed the tree to pluck a fruit'

POS-tagging Output:

naan	<PRP>
oru	<DET>
pazhaM	<NN>
paRikka	<VINT>
marattil	<NN>
eeRineen	<VF>
.	<DOT>

MST input format:

naan	oru	pazam	paRikka	marattil	eeRineen	.
PRP	DET	NN	VINT	NN	VF	.
0	0	0	0	0	0	0
0	0	0	0	0	0	0

MST output:

naan	oru	pazam	paRikka	marattil	eeRineen	.
PRP	DET	NN	VINT	NN	VF	.
N.SUB	DET	CL.DOBJ	XCL	X	ROOT	SYM
6	3	4	6	6	0	6

4.4. Dependency Tree Viewer

For viewing the dependency structures as tree we use dot Software i.e. Graphiz Tool. To change the MST output and MALT output into the input format of the Grapphiz Tool (Diagraph Format Conversion), two different Perl programs are used. The following is an example for MST parser tool.

MST output:

1	2	3	4	5	6	7
naan	oru	pazam	paRikka	marattil	eeRineen	.
PRP	DET	NN	VINT	NN	VF	.
NSUB	X	CLDOBJ	XCL	X	ROOT	SYM
6	3	4	6	6	0	6

Digraph Format conversion:

Conversion 1

1	naan	6	<N.SUB>
2	oru	3	<X>
3	pazam	4	<CL.DOBJ>
4	paRikka	6	<X.CL>
5	marattil	6	<X>
6	eeRineen	0	<ROOT>
7	.	6	<SYM>

Conversion 2

<N.SUB>(6_ eeRineen,1_ naan)

<X>(3_ pazam,2_ oru)

<CL.DOBJ>(4_ paRikka,3_ pazam)
<X.CL>(6_ eeRineen,4_ paRikka)
<X>(6_ eeRineen,5_ marattil)
<ROOT>(0_ROOT,6_ eeRineen)
<SYM>(6_ eeRineen,7_.)

Conversion 3 (Input of Graphiz Tool)

digraph 1 {
"1_ naan~"[label=" naan~"];
"6_ eeRineen" -> "1_ naan"[label="<NSUB>"];
"2_ oru "[label=" oru "];
"3_ pazam " -> "2_ oru "[label="<X>"];
"3_ pazam "[label=" pazam "];
"4_ paRikka" -> "3_ pazam"[label="<CLDOBJ>"];
"4_ paRikka"[label=" paRikka"];
"6_ eeRineen " -> "4_ paRikka"[label="<XCL>"];
"5_ marattil"[label=" marattil"];
"6_ eeRineen " -> "5_ marattil"[label="<X>"];
"6_ eeRineen "[label=" eeRineen "];
"0_ROOT" -> "6_ eeRineen "[label="<ROOT>"];
"7_."[label="."];
"6_ eeRineen " -> "7_."[label="<SYM>"];
}

The above data is the input format for the Graphiz Tool. An output is obtained using Graphiz Tool (dependency tree of the sentence). The dependency tree thus formed is similar to that given under MALT parser tool. The following the example for MALT Parser Tool.

MALT output:

1	naan	_	<PRP>	<PRP>	_	6	N.SUB	_	_
2	oru	_	<DET>	<DET>	_	3	X	_	_
3	pazam	_	<NN>	<NN>	_	4	CL.DOBJ	_	_
4	paRikka	_	<VINT>	<VINT>	_	6	X.CL	_	_
5	marattil	_	<NN>	<NN>	_	6	X	_	_
6	eeRineen	_	<VF>	<VF>	_	0	ROOT	_	_
7	.	_	<DOT>	<DOT>	_	6	SYM	_	_

Digraph Format conversion:

Conversion 1

1	naan	6	N.SUB
2	oru	3	X
3	pazam	4	CL.DOBJ
4	paRikka	6	X.CL
5	marattil	6	X
6	eeRineen	0	ROOT
7	.	6	SYM

Conversion 2

NSUB(6_ eeRineen,1_ naan)
DET(6_ eeRinee,2_ oru)
DOBJ(6_ eeRineen,3_ pazam)
VPCL(6_ eeRineen,4_ paRikka)
LOCMOD(6_ eeRineen,5_ marattil)
ROOT(0_ROOT,6_ eeRineen)
SYM(6_eeRineen,7_.)

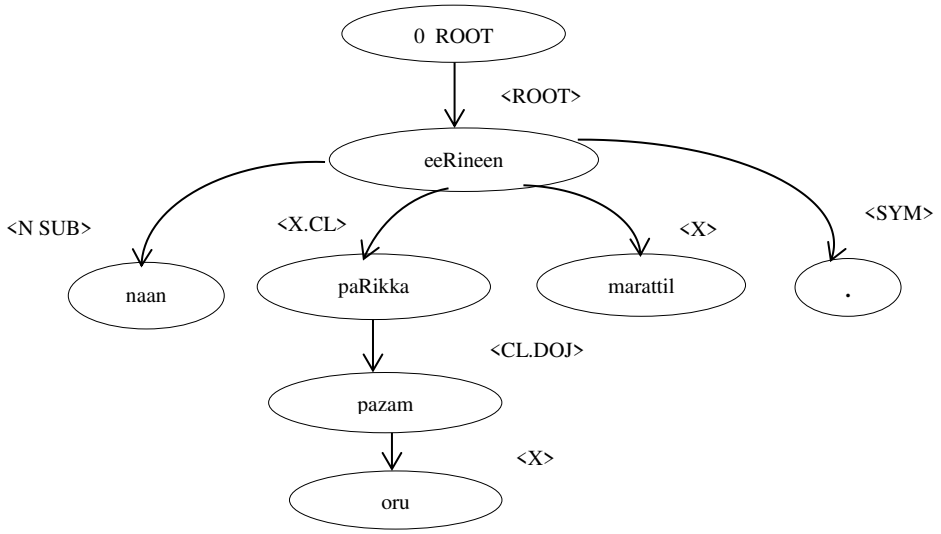
Conversion 3

naan oru paham paRikka marattil eeRineen

digraph 1 {
"1_ naan"[label="1_ naan"];

"6_ eeRineen" -> "1_ naan"[label="NSUB"];
"2_ oru "[label="2_ oru "];
"6_ eeRineen" -> "2_ oru "[label="X"];
"3_ pazam"[label="3_ pazam "];
"6_ eeRineen " -> "3_ pazam "[label="CLDOBJ"];
"4_ paRikka "[label="4_ paRikka~"];
"6_ eeRineen " -> "4_paRikka "[label="XCL"];
"5_ marattil"[label="5_ marattil~"];
"6_ eeRineen " -> "5_ marattil~"[label="X"];
"6_ eeRineen "[label="6_ kayaRi "];
"0_ROOT" -> "6_ kayaRi "[label="ROOT"];
"7_."[label="7_."];
"6_ eeRineen " -> "7_."[label="SYM"];
}

The above data is the input format for the Graphic Tool. Output using Graphic Tool is given below.



The machine learning approach requires huge training data as the accuracy of the output depends on the training data. We need to have sumptuous training data at the POS tagging level, chunking level and dependency parsing level. MALT parser is a data driven system for dependency parsing that can also be used for syntactic parsing. MALT parser generally achieves good parsing accuracy. MALT parser can achieve robust, efficient and accurate parsing for wide range of languages. MST parser tool is a language independent tool used for Dependency Parsing which is implemented in English language. Using these tools the dependency labels and the position of head of Malayalam language is obtained. The results of both the tools are encouraging. Evaluation of the Dependency Parsing is done using both MST and MALT parser Tools. Both give commendably good result. Moreover for short range Dependencies, MALT Parser Tool holds good.

5. Question answering system

Question Answering is a computer science discipline within the fields of information retrieval and natural language processing, which focuses on building systems that automatically answer questions posed by humans in a natural language. Valid answers mean answers relevant to the questions posed by the user. Stanford Question Answering Dataset (SQuAD) is a new reading comprehension dataset, consisting of questions posed by crowd workers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage. In information retrieval, an open domain question answering system aims at returning an answer in response to the user's question. The returned answer is in the form of short texts rather than a list of relevant documents. The system uses a combination of techniques from computational linguistics, information retrieval and knowledge representation for finding answers.

We are proposing a domain specific structured database from which the information related to Tourism can be retrieved. The questions will be parsed with the parser we have developed and the

answers will be extracted from the structured database by matching. We are yet to build the question answer system.

6. Conclusion

The dependency parser based on Stanford parser is very much needed for any kind of NLP tasks in Tamil. The dependency information is very helpful for finding the subject, object and indirect object and other arguments (participants) in a sentence. The dependency information is very useful for building a question-answer system for Tamil. The parser built by us works efficiently for Tamil. By augmenting the training corpus we can bring efficiency to the parser. The parser is used for parsing questions and parsing the 50 thousand sentences kept in the structured data base. Correlating the question with the sentences in the structured data base and getting the correct answer is the challenging one. We are yet to work on this direction.

BIBLIOGRAPHY

[1] Lehmann, T. 199. A Grammar of Modern Tamil. Pondicherry: Pondicherry Institute of Linguistics and Culture.

[2] Attardi, G. and Orletta, F.D. Chunking and Dependency Parsing. Technical paper.

[3]Attardi, G., Chanev, A., Ciaramita, M., Dell'Orletta, F. and Simi, M. 2007. Multilingual Dependency Parsing and Domain Adaptation using DeSR. Proceedings of the CoNLL Shared Task Session of of EMNLP-CoNLL, Prague.

[4]Buchholz, S., Marsi, E., Dubey, A. and Krymolowski, Y. 2006. Shared task on multilingual dependency parsing. In Proceedings of the Conference on Computational Natural Language Learning (CoNLL).

[5] Chang, C. and Lin, C. 2011. LIBSVM: A Library for Support Vector Machines. Technical Report, National Taiwan University, Taipei, Taiwan.

CRF Website http://crfpp.sourceforge.net.

[6] Dhanalakshmi V, Anand Kumar M., Rajendran S., and Soman K. P. 2009. POS Tagger and Chunker for Tamil Language. International Forum for Information Technology in Tamil.

[7] Dhanalakshmi, V., Anand Kumar, M., Loganathan R, Soman, K.P. and Rajendran S. 2008.Tamil part-of-Speech-Tagger based on SVM Tool. In proceeding of the COLIPS International conference on Asian Language Processing (IACP). Chang mai, Thailand.

[8] Dhanlakshmi,V. Shallow parser for Tamil. PhD thesis submitted to Tamil University, Thanjavur.

[9] Ghosh, A., Das, A., Bhaskar, P. and Bandyopadhyay, S. 2010. Bengali Parsing system. In Proceedings of International Conference On Natural Language Processing (ICON), Tool Contest.

[10] Gimenez J. and Marquez, L. 2006. SVMTool Technical Manual v1.3. Technical Manual. TALP Research Center, LSI Department, Universitat Politecnica de Catalunya, Barcelona.

[11] Hsu, C. Chang C. and Lin, C. 2010. A Practical Guide to Support Vector Classification. Technical Report, National Taiwan University, Taipei, Taiwan.

[12] Kesidi, S.R., Kosaraju, P., Vijay, M. and Husain, S. 2010. A two stage Constraint based Hybrid Dependency Parsing. Proceedings of International Conference On Natural Language Processing (ICON), Tool Contest.

[13] Kolachina, S., Kolachina, P., Agarwal, M. Husain, A. 2010. Experiments with MaltParser for parsing Indian Languages. Proceedings of International Conference On Natural Language Processing (ICON), Tool Contest.

[14] Kosaraju, P., Kesidi, S.R., Ainavolu, V.B.R. and Kukkadapu, P. 2010. Experiments on Indian Language Dependency Parsing. Proceedings of International Conference on Natural Language Processing (ICON) Tool Contest.

[15] Kubler, S., McDonald, R. Nivre, J. 2009. Dependency Parsing. Morgan & Claypool publishers.

[16] Lee, H. Park, S., Sang-Jo Lee, S. and Park, S. 2006. Korean Clause Boundary recognition. PRICAI'06 Proceedings of the 9th Pacific Rim international conference on Artificial intelligence. Springer-Verlag Berlin, Heidelberg.

[17] Kumara Shanmugam, B. 2004. Parse representation of Tamil syntax. Thesis submitted for Master of Science (by research), Department Of Electronics Engineering, Faculty Of Science And Humanities, Anna University. Chennai.

[18] MALT Parser website http://www.maltparser.org/userguide.html .

[19] Maneffe, M. De and Manning, C.D. 2008, 2016. Stanfod typed dependency manual. Revised for the Stanford parser v.3.7.0 in September 2016.
[20] McDonald, R. 2006. Discriminative Learning and Spanning Tree Algorithms for Dependency Parsing. Ph.D thesis. University of Pennsylvania.
[21] Nivre, J. 2005. Dependency Grammar and Dependency Parsing. Technical Report. School of Mathematics and Systems Engineering, Vaxjo University.
[22] Ohno, T., Matsubara, S., Kashioka, H., Kato, N. and Inagaki, Y. 2005. Incremental Dependency Parsing of Japanese Spoken Monologue Based on Clause Boundaries. Proceedings of 9th European Conference on Speech Communication and Technology (Interspeech-2005). Pages 3449-3452.
[23] Rajendran, S. 2006. Parsing in Tamil: Present State of Art. Indian Linguistics vol. 67, pages 159-67.
[24] Rekha R.U. 2010. Dependency Parsing for Tamil using Machine Learning Approaches. A Project Report. Amrita School of Engineering, Amrita University, Coimbatore.
[25] Sundar Ram, R.V. and Devi, S.L. 2008. Clause Boundary Identification using Conditional Random Fields. Springer, Heidelberg, pages 140-150.
[26] Titov, I. and Henderson, J. 2007. "A Latent Variable Model for Generative Dependency Parsing", Proceedings of International Conference on Parsing Technologies (IWPT -07), Prague.